People. Innovation. Excellence.

BINUS UNIVERSITY

30th Anniversary BINA NUSANTARA

BINUS ICTC PROSIDING 2011

BINUS ICT 2011 Conference

ICT for a Smarter World

Jakarta, August 2011

www.binus.ac.id | (021) 53 69 69 69

ISBN : 978-602-99817-0-4

ISBN : 978-602-99817-0-4

PENGGUNAAN INDEX UNTUK MENINGKATKAN PERFORMANCE PADA *ORACLE 10G - DATABASE*

Suparto Darudiato[1)], Handi Juseno[2)], Andre Sanjaya[3)], Ningsih[4)]

[1,2,3,4] Jurusan Sistem Informasi, Fakultas Ilmu Komputer, [5]BINUS University
Jl. KH. Syahdan No. 9 Palmerah Jakarta Barat
[1]supartod@binus.edu, [2]handi_2111@yahoo.com

Abstrak

Execution time yang cepat terhadap database, selalu menjadi tujuan dari semua pemakai aplikasi komputer. Banyak cara untuk membuat *excecution time* menjadi lebih cepat, salah satunya adalah sql tuning. Dalam penelitian ini, tidak hanya melakukan sql tuning, tetapi juga melakukan percobaan terhadap tabel yang belum normal dan dilanjutkan dengan tabel yang dinormalisasikan. Setelah tabel dinormalisasikan, tabel-tabel tersebut di buatkan index dan *primary key*. Hasilnya tabel yang sudah dinormalisasikan, ditambah dengan index mempunyai execution time yang jauh lebih cepat.

Kata kunci: *Execution Time, Sql, Turning, Tabel, Normalisasi*

Latar Belakang

Pada abad ke-21 ini, teknologi informasi mempunyai pengaruh yang besar dalam berbagai aspek kehidupan masyarakat, bahkan sudah merupakan bagian dari kehidupan masyarakat. Pengaruh tersebut sangat terasa pada bidang pendidikan, pemerintahan, kesehatan dan terutama di bidang bisnis. Karena semuanya membutuhkan informasi. Maka dalam era teknologi informasi yang serba cepat ini, perusahaan dituntut agar dapat tetap bertahan dan bersaing dengan perusahaan-perusahaan sejenis lainnya untuk kemajuan perusahaannya. Oleh sebab itu perusahaan harus bisa menerapkan dan memanfaatkan teknologi informasi secara efektif dan efisien dengan segala sumber daya yang ada.

Para pengguna aplikasi selalu menginginkan sebuah aplikasi serta *database* yang dapat mengurangi waktu mereka tetapi sebisa mungkin tidak perlu menambah infrastruktur yang ada, dan mudah untuk dikembangkan serta dimodifikasi sesuai dengan kebutuhan dan perkembangan proses bisnis yang ada. Untuk mengatasi hal tersebut ada beberapa hal yang bisa dilakukan. Salah satunya adalah melakukan *turning* terhadapa *database* yang ada.

Dengan melakukan turning terhadapa database, dapat memberikan mempercepat *execution time* pada saat memproses *query* yang dilakukan oleh *user* dan mampu mengoptimalkan penggunaan sumber daya yang ada tanpa menambah biaya baik *hardware* maupun *software*.

Metodologi

Untuk dapat menganalisis dengan baik masalah yang ada di dalam perusahaan dan mengusulkan alternatif pemecahan masalah yang tepat, maka perlu menggunakan beberapa metode penelitian. Berikut ini adalah beberapa metode yang digunakan:

a. Studi kepustakaan. Melakukan pencarian materi dan konsep yang berkaitan dengan topik melalui buku, internet serta jurnal Pengolahan Data
b. Menggunakan pendekatan Chan dan Ashdown [2], dalam hal:
   - Melakukan pemeriksaan standar awal *database*, dan aplikasi.
   - Memeriksa sepuluh kesalahan umum yang terdapat pada *database* Oracle, dan menentukan apakah mungkin menjadi masalah.
   - Membangun model konseptual tentang apa yang terjadi pada sistem.
   - Mengusulkan langkah-langkah pemecahan masalah untuk sistem, kemudian implementasikan.
   - Membuat validasi perubahan yang telah dibuat dengan hasil yang diinginkan, dan lihat apakah persepsi kinerja pengguna telah meningkat.

Tinjauan Pustaka

ISBN : 978-602-99817-0-4

Biasanya sebuah perusahaan menggunakan *database* oracle kalau transaksi diperusahaan sudah sangat besar dan membutuhkan tingkat keamanan serta akses yang tinggi. Untuk itu perlu pemahaman dasar cara kerja oracle dalam membaca *database* nya.

Pada level fisikal, Oracle membaca blok data. Jumlah terkecil data yang dibaca adalah satu blok Oracle, yang terbesar adalah dibatasi dengan limit sistem operasi (dan multiblok I/O). Secara logika, Oracle mencari data untuk dibaca menggunakan metode berikut *Full Table Scan* (FTS), *Index Lookup* (*unique* dan *non-unique*) dan *Rowid.*

Sampai saat ini, belum ada perangkat lunak ataupun database yang sempurna. Masing-masing mempunyai keunggulan dan kelemahannya masing-masing. Begitu juga dengan *database* oracle. Menurut Chan & Ashdown [2, 54] dalam database oracle terdapat sepuluh kesalahan umum, yaitu:

a. Manajemen koneksi yang kurang baik
b. Penggunaan *cursor* dan *shared pool* yang kurang baik
c. SQL yang kurang baik
d. Penggunaan inisialisasi parameter yang tidak standar akibat asumsi yang tidak benar.
e. I/O *database* yang salah
f. Masalah pada *setting online redo log*
g. Serialisasi dari blok-blok data pada *buffer cache* karena kekurangan dari *free list, free list group,*
h. *Long full table scans*
i. Banyaknya jumlah *recursive* (SYS) SQL
j. *Schema Errors* dan *Optimizer Problems*

SQL *Tuning*

Tuning adalah suatu tindakan untuk menemukan sesuatu yang menjadi penyebab masalah dan membuat suatu perbaikan yang diperlukan untuk mengurangi dampak dari masalah tersebut. Seperti yang disampaikan oleh Chan dan Ashdown [2, 22] bahwa *tuning* merupakan tindakan untuk mengidentifikasi hambatan-hambatan yang paling signifikan dan membuat perubahan yang diperlukan untuk mengurangi dampak dari permasalahan tersebut.

Menurut Connolly dan Begg [3, 558], manfaat atau keuntungan yang akan diperoleh jika melakukan *tuning* adalah sebagai berikut:

a. Dengan *tuning*, kita dapat meminimalkan pembelian *hardware* baru.
b. *Tuning* memungkinkan untuk menurunkan konfigurasi *hardware*. Hasilnya tidak terlalu banyak memerlukan *hardware* tambahan dan biaya penggunaan *hardware* yang lebih murah sehingga biaya pemeliharaannya juga lebih murah.
c. Sebuah sistem yang di-*tuning* dengan baik akan menghasilkan *response time* yang lebih cepat dan *throughput* yang lebih baik, sehingga membuat pemakai dan perusahaan menjadi lebih produktif.
d. Meningkatnya *response time* dapat meningkatkan semangat kerja dari para pegawai.
e. Meningkatnya *response time* dapat menambah tingkat kepuasan pelanggan.

SQL *tuning* merupakan tindakan mengoptimisasi *SQL statement* dan mengurangi *response time*. Dimana menurut Alapati [1], *query optimization* adalah suatu proses dalam memilih strategi eksekusi yang paling efisien dalam mengeksekusi sebuah *query*. Chan dan Ashdown [2, 384] mengatakan, ada tiga cara cara yang bisa dilakukan untuk meningkatkan performance, yitu Mengurangi *Workload*, Menyeimbangkan *Workload* dan Paralelisasi *Workload.*

Dalam SQL *statement* terdapat *Join Order*. *Join Order* dapat memberikan efek yang sangat signifikan pada *performance* akses data dalam *database. Hal yang perlu diperhatikan dalam menggunakan join order adalah :*

1. Hindarilah *full-table scan*, jika memungkinkan gunakan *index* untuk mendapatkan data yang dibutuhkan.
2. Hindarilah menggunakan *index* yang menarik 10.000 baris data dari sebuah tabel jika masih bisa menggunakan *index* lain yang menarik hanya 100 baris data.
3. Pilihlah *join order* yang tepat sehingga menghasilkan lebih sedikit baris untuk di-*join*-kan dengan tabel berikutnya.

Pada saat mengeksekusi suatu *query*, *optimizer* akan memilih *execution plan* berdasarkan *access path* yang tersedia. Jika terdapat lebih dari satu cara untuk mengeksekusi SQL *statement*, *optimizer* selalu memilih cara dengan rangking terendah. Selain *Join Order*, ada fasilitas yang disebut dengan *hint* dalam SQL *statement* yang berguna untuk menginstruksikan kepada SQL *optimizer* untuk menjalankan SQL *statement* sebagaimana mestinyanya.

***Hint* Oracle**

Hint adalah suatu sintaks yang disediakan Oracle untuk mempengaruhi *execution plan* sehingga bisa lebih cepat lagi kinerjanya dalam

ISBN : 978-602-99817-0-4

pengaksesan data jika *execution plan* tersebut *out of date*. Seperti apa yang disampaikan oleh Cunningham [4, 2], *hint* adalah suatu sintaks spesial dalam bentuk *comment* di dalam pernyataan SQL yang menspesifikasi suatu instruksi atau suatu *cost-based optimizer* (CBO). *Optimizer* akan menggunakan *hint* untuk mempengaruhi atau memaksa pilihannya dari suatu *execution plan*.

Hint dispesifikasikan dalam suatu pernyataan *block* di mana *delete, insert, select*, atau *update* merupakan *keyword query* yang memulai suatu pernyataan *block*. *Hint* memungkinkan untuk mempengaruhi *goal, behavior*, metode, dan pilihan yang dipilih *optimizer*. *Behavior optimizer* secara *default* ditentukan oleh informasi seperti statistik pada objek di dalam suatu *query*, sistem statistik, parameter-parameter *session* dan parameter *database*.

Biasanya *optimizer* akan memilih *query plan* yang telah didesain untuk menyediakan *throughput* dan *efisiensi* yang paling baik. Bagaimanapun, respon yang paling cepat mungkin menjadi *goal* yang lebih penting dibanding *throughput* untuk suatu aplikasi seperti *browser* yang dapat melakukan *scroll*.

Oracle *Indexes*

Seperti yang disampaikan oleh SageLogix (2009, p.2) *Index* adalah struktur data opsional yang berhubungan dengan tabel dan *cluster* yang digunakan untuk mempercepat akses *row* data dan kegunaan lainnya adalah dapat digunakan sebagai mekanisme untuk membuat nilai data lebih unik lagi dalam satu atau lebih kolom. Sedangkan menurut Alapati [1], *index* Oracle menyediakan akses yang cepat pada baris tabel dengan menyimpan nilai yang terurut dari kolom yang spesifik, dan menggunakan nilai yang terurut tersebut untuk melihat secara detail ke *row* tabel yang bersangkutan, hampir sama dengan cara menggunakan *index* buku untuk secara cepat menemukan apa yang diinginkan.

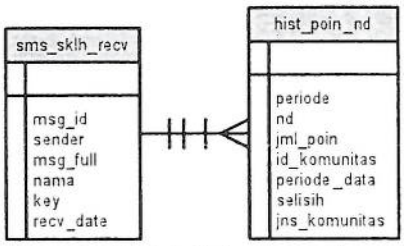
Index memungkinkan untuk mencari suatu *row* dengan nilai kolom tertentu tanpa harus mencari pada seluruh baris yang ada pada tabel tersebut tetapi cukup hanya dengan sebagian kecil baris yang ada di tabel tersebut. Dengan demikian, penggunaan *index* yang tepat akan mengurangi *disk* I/O yang mahal. Kegunaannya dari *index* yaitu meningkatkan waktu respon dalam pengambilan data untuk suatu *query*. Kemudian pada saat data berubah, *index* itu dibuat untuk mempertahankan keunikan nilai data dan mencegah nilai yang tidak unik masuk ke dalam tabel.

Index di Oracle terdiri dari tiga tipe yaitu *unique and nonunique indexes, primary and secondary indexes, dan composite indexes.*

Hasil dan Pembahasan

Berdasarkan table yang berada pada gambar 4.1, dilakukan uji coba dengan menggunakan query seperti pada tabel 4.1. Proses eksekusi dari query tersebut ditampilkan pada gambar 4.2 dan hasilnya ditampilkan pada tabel 4.2.

Berdasarkan uji coba *query* pada tabel 4.1 terlihat *exection time* yang diperlukan sebesar 6.2 detik (lihat gambar 4.2). Hal ini terjadi dikarenakan *query* ini mengalami *full table scan*, dengan kata lain tidak menggunakan *index*. Dan berdasarkan pada TKPROF yang ditunjukan pada tabel 4.2, menjelaskan bahwa penggunaan CPU sebesar 1,35 detik, pembacaan *disk* pada *datafile* sebesar 34.856 blok dan *consistent get* sebesar 34.877 *buffer* pada saat melakukan *parse*, *execute* dan *fetch* secara keseluruhan.



Gambar 4.1. Contoh Tabel yang digunakan pada aplikasi det_ps.php

Tabel 4.1. *Query* yang terdapat pada *file* det_ps.php untuk tabel yg belum dinormalisasi

```
SELECT '0'||SUBSTR(a.ND,-10) MDN, a.selisih,
b.nama
FROM HIST_POIN_ND a, SMS_SKLH_RECV b
WHERE id_komunitas = 'albana' AND jns_komunitas
= 'COOL' AND ND = sender
ORDER BY selisih desc
```

Gambar 4.2 Query viewer dan explain plan query pada file det_ps.php

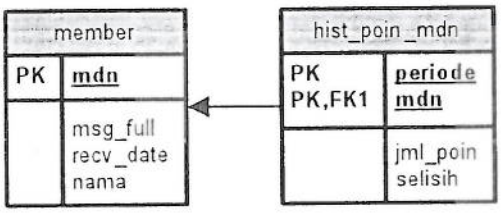
ISBN : 978-602-99817-0-4

Tabel 4.2. TKPROF dari *query* pada *file* det_ps.php

call	count	cpu	elaps	disk	Query	curr	rows
Parse	1	0.00	0.00	0	0	0	0
Execute		0.00	0.00	0	0	0	0
Fetch	1 3	1.35	7.32	34856	34877	0	31
total	5	1.35	7.33	34856	34877	0	31

Proses *query* yang dilakukan berdasarkan tabel 4.1, selain *full table scan* kedua table tersebutpun masih dalam kondisi tidak ternormalisasi. Secara konsep, harusnya table yang tidak normal akan mempunyai akses time yang relatif cepat. Untuk membukti hal tersebut, maka table pada gambar 4.1, dilakukan normalisasi sehingga menjadi seperti pada gambar 4.3.

Tentu saja, selain dilakukan normalisasi, dilakukan juga pembuangan terhadap atribut yang tidak dibutuhkan oleh aplikasi.



Gambar 4.3. Contoh tabel yang sudah di Normalisasi

Gambar 4.3 diatas merupakan table yang dinormalisasi berdasarkan gambar 4.1 diatas. Dimana awalnya kedua table pada gambar 4.1. tidak menggunakan primary key dan foreign key. Setelah di normalisasi, kedua jenis key tersebut muncul.

Hasil dari normalisasi yang dilakukan mengacu kepada dua syarat, yaitu :

- Hasil dari query harus sama dengan hasil query seperti sintaks query pada table 4.1.
- Kecepatan dari akses query tersebut harus lebih cepat dibandingkan dengan akses time pada table 4.1.

Query pada tabel 4.3 berikut ini adalah *query* yang telah mengalami perubahaan dalam menghasilkan data yang sama dengan *query* pada sistem lama (tabel 4.1).

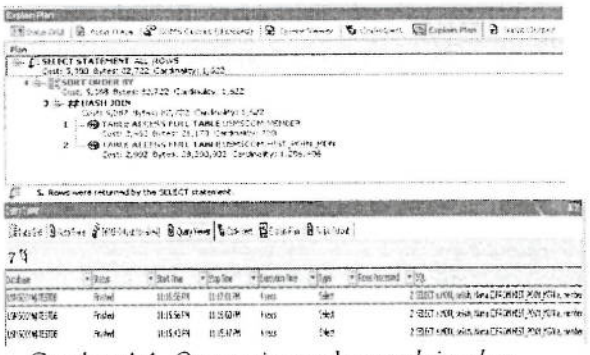
Tabel 4.3 *Query* yang terdapat pada *file* det_ps.php utk tabel normalisasi

```
SELECT h.MDN, selisih, Nama FROM
HIST_POIN_MDN h, member m where m.mdn
= h.mdn and m.id_komunitas = 'albana'
and m.id_cat_kom = '1' and periode
='201010' order by selisih desc
```

Dalam melakukan *tuning*, ada bererapa hal yang bisa dilakukan. Dalam kesempatan ini, tuning yang dilakukan ada sql tuning. Berikut ini adalah tahapan dalam melakukan *tuning* pada *file* det_ps.php:

a. Query Tabel Normalisasi

Apabila *Query* pada tabel 4.3 dijalankan, akan mengalami *full table scan*, hal ini disebabkan karena proses pencarian data dilakukan pada semua data yang terdapat pada masing-masing tabel. Berikut ini adalah gambar *explain plan* dan *query viewer* dari eksekusi *query* tersebut:



Gambar 4.4. *Query viewer* dan *explain plan query* pada *file* det_ps.php pada tabel normalisasi

Tabel 4.4. TKPROF pada *file* det_ps.php yang sudah di normalisasi

call	count	cpu	elaps	disk	Query	curr	rows
Parse	1	0.00	0.00	0	0	0	0
Execute		0.00	0.00	0	0	0	0
Fetch	1 2	1.53	4.01	20757	23532	0	2
total	4	1.53	4.01	20757	23532	0	2

Berdasarkan *query* pada tabel 4.3, maka *execution time* yang diperlukan sebesar empat detik (lihat gambar 4.4.). Dan berdasarkan pada TKPROF yang ditunjukan pada tabel 4.4, terlihat bahwa penggunaan CPU sebesar 1,53 detik, pembacaan *disk* pada *datafile* sebesar 20.757 blok dan *consistent get* sebesar 23.532 *buffer* pada saat melakukan *parse*, *execute* dan *fetch* secara keseluruhan.

b. Pemberian *index* pada *primary key*

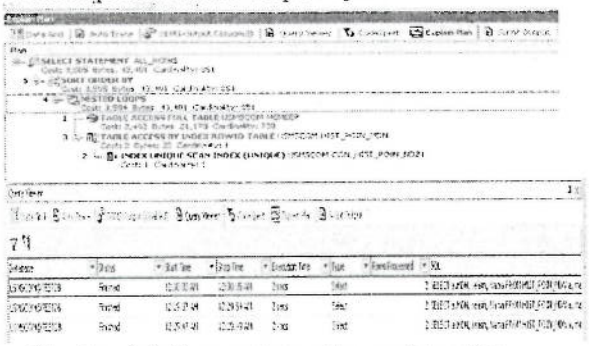
Tahap kedua adalah pemberian *index* pada *primary key* di semua tabel normalisasi dan dilanjutkan dengan menjalankan query pada tabel 4.3, maka terlihat gambar 4.5 dibawah ini, bahwa proses pencarian data pada tabel

ISBN : 978-602-99817-0-4

hist_poin_mdn dilakukan dengan menggunakan *index* pada *primary key* sedangkan pada tabel member proses pencarian dilakukan pada seluruh data dalam tabel member. Proses pencarian dilakukan pada seluruh data dalam tabel member disebabkan karena terdapat proses penyaringan (*where clause*) yang tidak menggunakan *primary key* pada tabel member dan tidak adanya nilai pencarian yang tetap pada *field* mdn (*primary key*) pada tabel member.

Hasil dari query pada tabel 4.3, terlihat *execution time* yang diperlukan setelah penambahan *index* pada *primary key* yaitu sebesar 2,26 detik (lihat tabel 4.5). Dan berdasarkan pada TKPROF yang ditunjukan pada tabel 4.5, menjelaskan bahwa penggunaan CPU sebesar 0,39 detik, pembacaan *disk* pada *datafile* sebesar 10.870 blok dan *consistent get* sebesar 11.246 *buffer* pada saat melakukan *parse*, *execute* dan *fetch* secara keseluruhan. Hasil dari pembacaan *disk* dan penggunaan *buffer* ini menjadi lebih sedikit dibandingkan dengan poin sebelumnya karena adanya penggunaan *index* pada tabel hist_poin_mdn langsung menuju *rowid* yang dicari.

Berdasarkan penjelasan pada *execution time* yang diperlukan mencapai dua detik disebabkan adanya proses *full table scan* dan penggunaan *index* pada tabel hist_poin_mdn ketika proses *nested loops* terjadi.



Gambar 4.5 *Query viewer* dan *explain plan query* pada *file* det_ps.php setelah menggunakan *index* pada *primary key*

Tabel 4.5. TKPROF pada *file* det_ps.php setelah menggunakan *index* pada *primary key*

call	count	cpu	elaps	disk	Query	curr	Rows
Parse	1	0.00	0.00	0	0	0	0
Execute		0.00	0.00	0	0	0	0
Fetch	1	0.39	2.26	10870	11246	0	2
	2						
total	4	0.39	2.26	10870	11246	0	2

c. Pemberian *index composite*

Pada tahap sebelumnya, terdapat proses *full table scan* pada tabel member yang menyebabkan proses menjadi lambat. Untuk mengatasi hal tersbut, diberikan *index composite* pada kedua *field* dalam tabel member (id_komunitas, id_cat_kom), karena kedua *field* tersebut digunakan di dalam *where clause* pada *query* dalam tabel 4.3. Pemberian *index* pada *field* id_komunitas dan id_cat_kom selain digunakan pada *query* dalam tabel 4.3. Tabel 4.6 berikut ini merupakan sintaks dari pembuatan *index* tersebut:

Tabel 4.6. *Index Composite* yang digunakan pada tabel member

```
create index idx_member_composite on
member(id_komunitas, id_cat_kom)
```

Untuk pembuktian execution time terhadap tabel member yang telah dibuatkan index composite-nya, tetap digunakan query yang terdapat pada tabel 4.3.

Gambar 4.6 dibawah menunjukkan query viewer dan explain plan query dari percobaan query berdasarkan tabel 4.3. Dari gambar 4.6 tersebut, terlihat *execution time* yang diperlukan tidak mencapai satu detik. Hal ini dikarenakan adanya penambahan *index composite* pada tabel member sehingga proses pencarian data tidak dilakukan pada keseluruhan tabel. Dan berdasarkan pada TKPROF yang ditunjukan pada tabel 4.7, menjelaskan bahwa penggunaan CPU sebesar 0,03 detik, pembacaan *disk* pada *datafile* sebesar 22 blok dan *consistent get* sebesar 397 *buffer* pada saat melakukan *parse*, *execute* dan *fetch* secara keseluruhan. Hasil dari pembacaan *disk* dan penggunaan *buffer* ini menjadi lebih sedikit dibandingkan dengan poin sebelumnya karena adanya penggunaan *index composite* yang telah dibuat pada tabel 4.6.

ISBN : 978-602-99817-0-4

Gambar 4.6 *Query viewer* dan *explain plan query* pada *file* det_ps.php setelah menggunakan *index*

Tabel 4.7 TKPROF pada *file* det_ps.php setelah menggunakan *index* pada *primary key* dan *index composite*

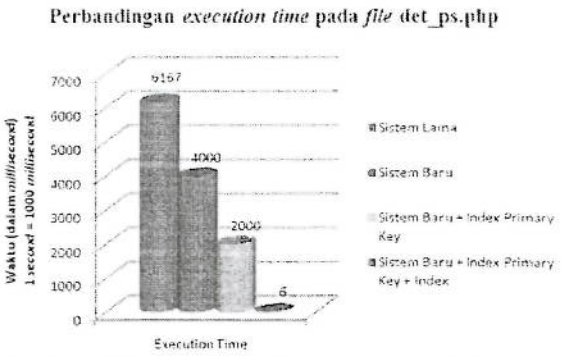
call	count	cpu	elaps	disk	Query	curr	Rows
Parse	1	0.00	0.00	0	0	0	0
Execute		0.00	0.00	0	0	0	0
Fetch	1	0.03	0.13	22	397	0	2
	2						
total	4	0.03	0.13	22	397	0	2

Dari hasil percobaan SQL *query* yang dilakukan sebanyak tiga kali pada *file* det_ps.php, maka dapat dilihat bahwa proses *execution time* pada struktur *database* yang telah dinormalisasi mengalami peningkatan. Tabel 4.8 berikut ini merupakan hasil perbandingan *execution time* yang dihasilkan sistem lama dengan sistem baru yang dilakukan secara bertahap:

Tabel 4.8 *Execution time query* pada sistem lama dan sistem baru dalam *file* det_ps.php

File **det_ps.php**	**Perc. 1** ***(msecond)***	**Perc. 2** ***(msecond)***	**Perc. 3** ***(msecond)***	**Rata2** ***(msecond)***
Sistem Lama	6200	6100	6200	**6167**
Sistem Baru	4000	4000	4000	**4000**
Sistem Baru + *index primary key*	2000	2000	2000	**2000**
Sistem Baru + *index*	16	1	1	**6**

Berdasarkan tabel 4.8, dibuat grafik seperti pada gambar 4.7. Terlihat jelas perbedaan dari sistem lama dengan sistem baru. Sistem baru pun dibagi menjadi tiga tahap. Dari hasil perbandingan pada gambar 4.7 terlihat bahwa setelah dilakukan percobaan sebanyak tiga kali, rata-rata *execution time* pada sistem baru menjadi lebih cepat yaitu selama enam milidetik (meningkat sebesar 102.783%). Hal ini disebabkan karena menggunakan *index* berupa *primary key* yang telah dibuat pada saat melakukan normalisasi dan *index composite*.



Gambar 4.7 Grafik *execution time* pada sistem lama dan sistem baru dalam *file* det_ps.php

Simpulan

Berdasarkan analisis, implementasi, dan evaluasi yang telah dilakukan, dapat ditarik beberapa simpulan sebagai berikut:

- Dengan menggunakan struktur *database* normalisasi, *data storage* yang digunakan menjadi lebih hemat dari segi penyimpanan datanya.
- Dengan dilakukan perbaikan *query* yang ada pada file det_ps.php mengakibatkan aplikasi ini menjadi lebih baik.
- Setelah dilakukan SQL *Tuning* dengan cara mengubah *query* menjadi lebih optimal serta ditambahkan *index*, kinerja aplikasi det_ps.php menjadi lebih cepat sehingga dapat membantu *user* dalam menyelesaikan pekerjaannya.

Daftar Pustaka

[1] Alapati, S. R. (2005). *Expert Oracle Database 10g Administration.* New York: Apress.

[2] Chan, I., & Ashdown, L. (2009). *Performance Tuning Guide 11g released 2(11.2).* USA: Oracle Corporation.

[3] Connolly, T., & Begg, C. (2010). *Database Systems A Practical Approach to Design, Implementation, and Management* (5th ed.). Boston: Pearson Education.

[4] Cunningham, R. (2009, 06 16). *Using Query Hints for Top SQL Performance.* Retrieved 06 16, 2011, from http://www.sagelogix.com/idc/groups/public/documents/sagelogix-whitepaper/sage015061.pdf: http://www.sagelogix.com

[5] Oracle. (2002). *Oracle9i Database Performance Tuning Guide and*

ISBN : 978-602-99817-0-4

Reference. Retrieved 06 15, 2011, from http://download.oracle.com/docs/cd/B10501_01/server.920/a96533/rbo.htm: http://oracle.com

[6] Oracle. (2005). *SQL*Plus® User's Guide and Reference*. Retrieved 06 15, 2011, from http://download.oracle.com/docs/cd/B19306_01/server.102/b14357/ch8.htm: http://oracle.com

[7] SageLogix. (2009, 11 12). *Understanding Indexes*. Retrieved 06 16, 2011, from http://www.sagelogix.com/idc/groups/public/documents/sagelogix-whitepaper/sage016101.pdf: http://www.sagelogix.com